A Tractable Extension of Linear Indexed Grammars

Bill Keller and **David Weir**
School of Cognitive and Computing Sciences
University of Sussex
Falmer, Brighton BN1 9QH
UK
bill.keller/david.weir@cogs.sussex.ac.uk

Abstract

Vijay-Shanker and Weir (1993) show that Linear Indexed Grammars (LIG) can be processed in polynomial time by exploiting constraints which make possible the extensive use of structure-sharing. This paper describes a formalism that is more powerful than LIG, but which can also be processed in polynomial time using similar techniques. The formalism, which we refer to as Partially Linear PATR (PLPATR) manipulates feature structures rather than stacks.

1 Introduction

Unification-based grammar formalisms can be viewed as generalizations of Context-Free Grammars (CFG) where the nonterminal symbols are replaced by an infinite domain of feature structures. Much of their popularity stems from the way in which syntactic generalization may be elegantly stated by means of constraints amongst features and their values. Unfortunately, the expressivity of these formalisms can have undesirable consequences for their processing. In naive implementations of unification grammar parsers, feature structures play the same role as nonterminals in standard context-free grammar parsers. Potentially large feature structures are stored at intermediate steps in the computation, so that the space requirements of the algorithm are expensive. Furthermore, the need to perform non-destructive unification means that a large proportion of the processing time is spent copying feature structures.

One approach to this problem is to refine parsing algorithms by developing techniques such as restrictions, structure-sharing, and lazy unification that reduce the amount of structure that is stored and hence the need for copying of features structures (Shieber, 1985; Pereira, 1985; Karttunen and Kay, 1985; Wroblewski, 1987; Gerdemann, 1989; Godden, 1990; Kogure, 1990; Emele, 1991; Tomabechi, 1991; Harrison and Ellison, 1992)). While these techniques can yield significant improvements in performance, the generality of unification-based grammar formalisms means that there are still cases where expensive processing is unavoidable. This approach does not address the fundamental issue of the tradeoff between the descriptive capacity of a formalism and its computational power.

In this paper we identify a set of constraints that can be placed on unification-based grammar formalisms in order to guarantee the existence of polynomial time parsing algorithms. Our choice of constraints is motivated by showing how they generalize constraints inherent in Linear Indexed Grammar (LIG). We begin by describing how constraints inherent in LIG admit tractable processing algorithms and then consider how these constraints can be generalized to a formalism that manipulates trees rather than stacks. The constraints that we identify for the tree-based system can be regarded equally well as constraints on unification-based grammar formalisms such as PATR (Shieber, 1984).

2 From Stacks to Trees

An Indexed Grammar (IG) can be viewed as a CFG in which each nonterminal is associated with a stack of indices. Productions specify not only how nonterminals can be rewritten but also how their associated stacks are modified. LIG, which were first described by Gazdar (1988), are constrained such that stacks are passed from the mother to at most a single daughter.

For LIG, the size of the domain of nonterminals and associated stacks (the analogue of the nonterminals in CFG) is not bound by the grammar. However, Vijay-Shanker and Weir (1993) demonstrate

that polynomial time performance can be achieved through the use of structure-sharing made possible by constraints in the way that LIG use stacks. Although stacks of unbounded size can arise during a derivation, it is not possible for a LIG to specify that two dependent, unbounded stacks must appear at distinct places in the derivation tree. Structure-sharing can therefore be used effectively because checking the applicability of rules at each step in the derivation involves the comparison of structures of limited size.

Our goal is to generalize the constraints inherent in LIG to a formalism that manipulates feature structures rather than stacks. As a guiding heuristic we will avoid formalisms that generate tree sets with an *unbounded* number of unbounded, dependent branches. It appears that the structure-sharing techniques used with LIG cannot be generalized in a straightforward way to such formalisms.

Suppose that we generalize LIG to allow the stack to be passed from the mother to two daughters. If this is done recursion can be used to produce an unbounded number of unbounded, dependent branches. An alternative is to allow an unbounded stack to be shared between two (or more) daughters but *not* with the mother. Thus, rules may mention more than one unbounded stack, but the stack associated with the mother is still associated with at most one daughter. We refer to this extension as Partially Linear Indexed Grammars (PLIG).

Example 1 *The* PLIG *with the following productions generates the language*

$$\{\, a^n b^m c^n d^m \mid n, m \geq 1 \,\}$$

and the tree set shown in Figure 1. Because a single PLIG *production may mention more than one unbounded stack, variables (x, y) are introduced to distinguish between them. The notation $A[x\sigma]$ is used to denote the nonterminal A associated with any stack whose top symbol is σ.*

$$\begin{array}{ll} A[x] \rightarrow aA[x\sigma], & A[x] \rightarrow B[y]C[x]D[y], \\ B[x\sigma] \rightarrow bB[x], & B[\sigma] \rightarrow b, \\ C[x\sigma] \rightarrow cC[x], & C[\sigma] \rightarrow c, \\ D[x\sigma] \rightarrow dD[x], & D[\sigma] \rightarrow d. \end{array}$$

Example 2 *A* PLIG *with the following productions generates the k-copy language over $\{ a, b \}^*$, i.e., the language*

$$\{\, w^k \mid w \in \{ a, b \}^* \,\}$$

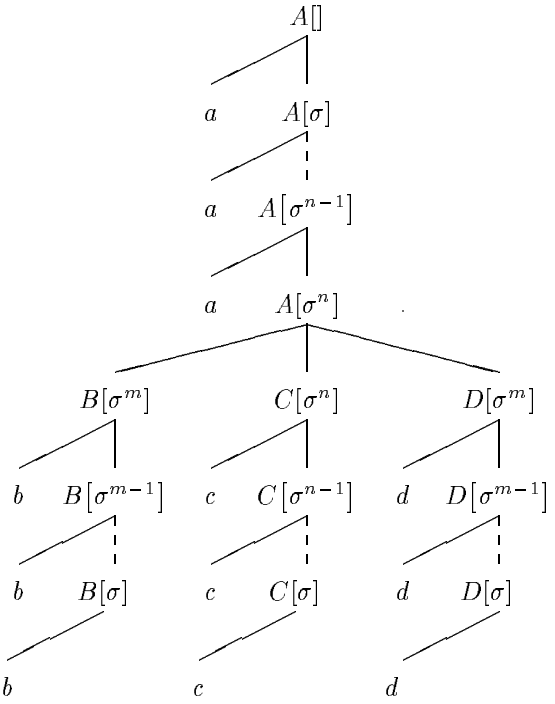


Figure 1: Tree set for $\{\, a^n b^m c^n d^m \mid n, m \geq 1 \,\}$

where $k \geq 1$.

$$S[] \rightarrow \underbrace{A[x] \ldots A[x]}_{k \text{ copies}}, \quad A[] \rightarrow \lambda,$$

$$A[x\sigma_1] \rightarrow a\, A[x], \qquad A[x\sigma_2] \rightarrow b\, A[x].$$

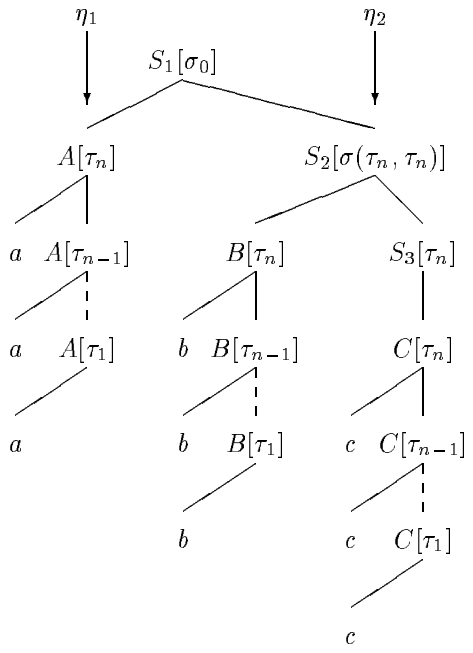
Example 3 PLIG *can "count" to any fixed k, i.e., a* PLIG *with the following productions generates the language*

$$\{\, a_1^n \ldots a_k^n \mid n \geq 0 \,\}$$

where $k \geq 1$.

$$\begin{array}{ll} S[] \rightarrow A_1[x] \ldots A_k[x], & \\ A_1[x\sigma] \rightarrow a_1\, A_1[x], & A_1[] \rightarrow \lambda, \\ \vdots & \\ A_k[x\sigma] \rightarrow a_k\, A_k[x], & A_k[] \rightarrow \lambda. \end{array}$$

In PLIG, stacks shared amongst siblings cannot be passed to the mother. As a consequence, there is no possibility that recursion can be used to increase the number of dependent branches. In fact, the number of dependent branches is bounded by the length of the right-hand-side of productions. By the same token, however, PLIG may only generate structural



where $\tau_1 = \sigma_1$ and $\tau_{i+1} = \sigma_2(\tau_i)$

Figure 2: Tree set for $\{ a^n b^n c^n \mid n \geq 1 \}$

descriptions in which dependent branches begin at nodes that are siblings of one another. Note that the tree shown in Figure 2 is unobtainable because the branch rooted at η_1 is dependent on more than one of the branches originating at its sibling η_2.

This limitation can be overcome by moving to a formalism that manipulates trees rather than stacks. We consider an extension of CFG in which each nonterminal A is associated with a tree τ. Productions now specify how the tree associated with the mother is related to the trees associated with the daughters. We denote trees with first order terms. For example, the following production requires that the x and y subtrees of the mother's tree are shared with the B and C daughters, respectively. In addition, the daughters have in common the subtree z.

$$A[\sigma_0(x, y)] \rightarrow \begin{array}{l} B[\sigma_1(x, z)] \\ C[\sigma_2(y, z)] \end{array}$$

There is a need to incorporate some kind of generalized notion of linearity into such a system. Corresponding to the *linearity* restriction in LIG we require that any part of the mother's tree is passed to at most one daughter. Corresponding to the *partial* linearity of PLIG, we permit subtrees that are not shared with the mother to be shared amongst the daughters. Under these conditions, the tree set

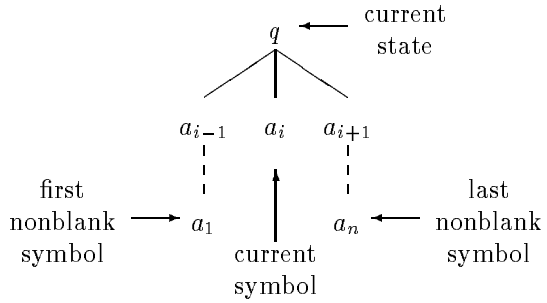


Figure 3: Encoding a Turing Machine

shown in Figure 2 can be generated. The nodes η_1 and η_2 share the tree τ_n, which occurs twice at the node η_2. At η_2 the two copies of τ_n are distributed across the daughters.

The formalism as currently described can be used to simulate arbitrary Turing Machine computations. To see this, note that an instantaneous description of a Turing Machine can be encoded with a tree as shown in Figure 3. Moves of the Turing Machine can be simulated by unary productions. The following production may be glossed: "if in state q and scanning the symbol X, then change state to q', write the symbol Y and move left" [1].

$$A[q(W(x), X, y)] \rightarrow A[q'(x, W, Y(y))]$$

One solution to this problem is to prevent a single daughter sharing more than one of its subtrees with the mother. However, we do not impose this restriction because it still leaves open the possibility of generating trees in which every branch has the same length, thus violating the condition that trees have at most a bounded number of unbounded, dependent branches. Figure 4 shows how a set of such trees can be generated by illustrating the effect of the following production.

$$A[\sigma(\sigma(x, y), \sigma(x', y'))] \rightarrow \begin{array}{l} A[\sigma(z, x)] \\ A[\sigma(z, y)] \\ A[\sigma(z, x')] \\ A[\sigma(z, y')] \end{array}$$

To see this, assume (by induction) that all four of the daughter nonterminals are associated with the full binary tree of height i (τ_i). All four of these trees are constrained to be equal by the production given above, which requires that they have identical left (i.e. z) subtrees (these subtrees must be the full binary tree τ_{i-1}). Passing the right subtrees (x, y, x' and y') to the mother as shown allows the

[1] There will be a set of such productions for each tape symbol W.

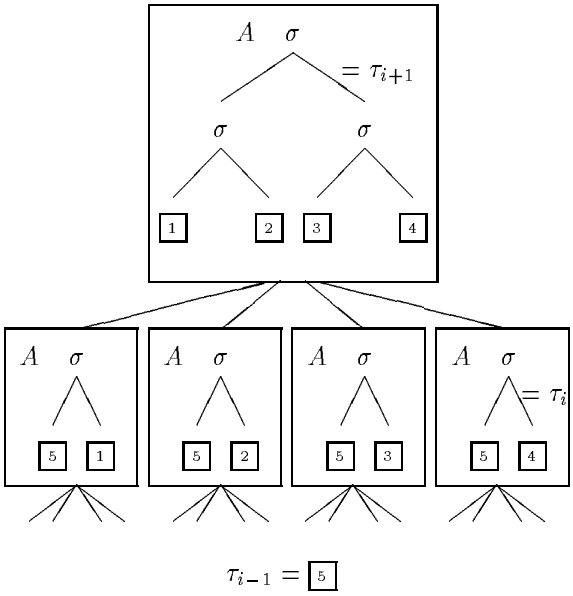


Figure 4: Building full binary trees

construction of a full binary tree with height $i+1$ (τ_{i+1}). This can be repeated an unbounded number of times so that all full binary trees are produced.

To overcome both of these problems we impose the following additional constraint on the productions of a grammar. We require that subtrees of the mother that are passed to daughters that share subtrees with one another must appear as siblings in the mother's tree. Note that this condition rules out the production responsible for building full binary trees since the x, y, x' and y' subtrees are not siblings in the mother's tree despite the fact that all of the daughters share a common subtree z. Moreover, since a daughter shares subtrees with itself, a special case of the condition is that subtrees occurring within some daughter can only appear as siblings in the mother. This condition also rules out the Turing Machine simulation. We refer to this formalism as Partially Linear Tree Grammars (PLTG). As a further illustration of the constraints places on shared subtrees, Figure 5 shows a local tree that could appear in a derivation tree. This local tree is licensed by the following production which respects all of the constraints on PLTG productions.

$$\begin{aligned} A[\sigma_1(\sigma_2(x_1, x_2, x_3), \sigma_3(x_4, \sigma_4))] \rightarrow \\ \quad B[\sigma_5(x_5, x_5, x_1)] \\ \quad C[\sigma_6(\sigma_7, x_4)] \\ \quad D[\sigma_8(x_2, x_3, x_5)] \end{aligned}$$

Note that in Figure 5 the daughter nodes labelled B and D share a common subtree and the subtrees

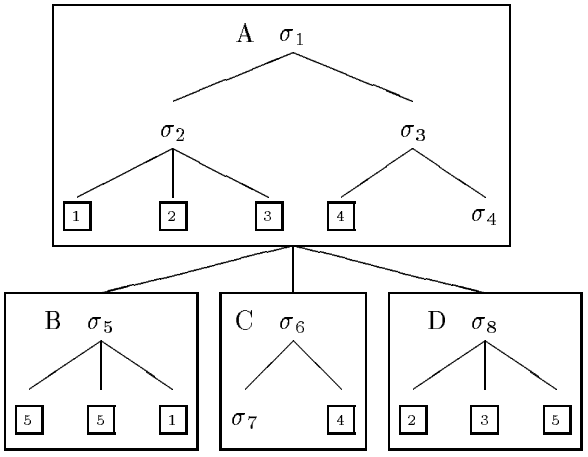


Figure 5: A PLTG local tree

shared between the mother and the B and D daughters appear as siblings in the tree associated with the mother.

Example 4 *The* PLTG *with the following productions generates the language*

$$\{\, a^n b^n c^n \mid n \geq 1 \,\}$$

and the tree set shown in Figure 2.

$$\begin{aligned} &S_1[\sigma_0] \rightarrow A[x]\, S_2[\sigma(x,x)], \\ &S_2[\sigma(x,y)] \rightarrow B[x]\, S_3[y], \\ &S_3[x] \rightarrow C[x], \\ &A[\sigma_2(x)] \rightarrow aA[x], \qquad A[\sigma_1] \rightarrow a, \\ &B[\sigma_2(x)] \rightarrow bB[x], \qquad B[\sigma_1] \rightarrow b, \\ &C[\sigma_2(x)] \rightarrow cC[x], \qquad C[\sigma_1] \rightarrow c. \end{aligned}$$

Example 5 *The* PLTG *with the following productions generates the language of strings consisting of k copies of strings of matching parenthesis, i.e., the language*

$$\{\, w^k \mid w \in D \,\}$$

where $k \geq 1$ and D is the set of strings in $\{(,)\}^*$ that have balanced brackets, i.e, the Dyck language over $\{(,)\}$.

$$S[] \rightarrow \underbrace{A[x] \ldots A[x]}_{k \text{ copies}}, \quad A[] \rightarrow \lambda,$$

$$A[\sigma_1(x)] \rightarrow (\, A[x] \,), \quad A[\sigma_2(x,y)] \rightarrow A[x]\, A[y].$$

3 Trees to Feature Structures

Finally, we note that acyclic feature structures without re-entrancy can be viewed as trees with branches labelled by feature names and atomic values only found at leaf nodes (interior nodes being unlabelled). Based on this observation, we can consider the constraints we have formulated for the tree system PLTG as constraints on a unification-based grammar formalism such as PATR. We will call this system Partially Linear PATR (PLPATR). Having made the move from trees to feature structures, we consider the possibility of re-entrancy in PLPATR.

Note that the feature structure at the root of a PLPATR derivation tree will not involve re-entrancy. However, for the following reasons we believe that this does not constitute as great a limitation as it might appear. In unification-based grammar, the feature structure associated with the root of the tree is often regarded as the structure that has been derived from the input (i.e., the output of a parser). As a consequence there is a tendency to use the grammar rules to accumulate a single, large feature structure giving a complete encoding of the analysis. To do this, unbounded feature information is passed up the tree in a way that violates the constraints developed in this paper. Rather than giving such prominence to the root feature structure, we suggest that the entire derivation tree should be seen as the object that is derived from the input, i.e., this is what the parser returns. Because feature structures associated with all nodes in the tree are available, feature information need only be passed up the tree when it is required in order to establish dependencies within the derivation tree. When this approach is taken, there may be less need for re-entrancy in the root feature structure. Furthermore, re-entrancy in the form of shared feature structures within and across nodes will be found in PLPATR (see for example Figure 5).

4 Generative Capacity

LIG are more powerful than CFG and are known to be weakly equivalent to Tree Adjoining Grammar, Combinatory Categorial Grammar, and Head Grammar (Vijay-Shanker and Weir, 1994). PLIG are more powerful than LIG since they can generate the k-copy language for any fixed k (see Example 2). Slightly more generally, PLIG can generate the language

$$\{ w^k \mid w \in R \}$$

for any $k \geq 1$ and regular language R. We believe that the language involving copies of strings of matching brackets described in Example 5 cannot be generated by PLIG but, as shown in Example 5, it can be generated by PLTG and therefore PLPATR. Slightly more generally, PLTG can generate the language

$$\{ w^k \mid w \in L \}$$

for any $k \geq 1$ and context-free language L. It appears that the class of languages generated by PLTG is included in those languages generated by Linear Context-Free Rewriting Systems (Vijay-Shanker et al., 1987) since the construction involved in a proof of this underlies the recognition algorithm discussed in the next section.

As is the case for the tree sets of IG, LIG and Tree Adjoining Grammar, the tree sets generated by PLTG have path sets that are context-free languages. In other words, the set of all strings labelling root to frontier paths of derivation trees is a context-free language. While the tree sets of LIG and Tree Adjoining Grammars have independent branches, PLTG tree sets exhibit dependent branches, where the number of dependent branches in any tree is bounded by the grammar. Note that the number of dependent branches in the tree sets of IG is not bounded by the grammar (e.g., they generate sets of all full binary trees).

5 Tractable Recognition

In this section we outline the main ideas underlying a polynomial time recognition algorithm for PLPATR that generalizes the CKY algorithm (Kasami, 1965; Younger, 1967). The key to this algorithm is the use of structure sharing techniques similar to those used to process LIG efficiently (Vijay-Shanker and Weir, 1993). To understand how these techniques are applied in the case of PLPATR, it is therefore helpful to consider first the somewhat simpler case of LIG.

The CKY algorithm is a bottom-up recognition algorithm for CFG. For a given grammar G and input string $a_1 \ldots a_n$ the algorithm constructs an array P, having n^2 elements, where element $P[i,j]$ stores all and only those nonterminals of G that derive the substring $a_i \ldots a_j$. A naive adaptation of this algorithm for LIG recognition would involve storing a set of nonterminals and their associated stacks. But since stack length is at least proportional to the length of the input string, the resultant algorithm would exhibit exponential space and time complexity in the worst case. Vijay-Shanker and Weir (1993) showed that the behaviour of the naive algorithm can be improved upon. In LIG derivations the application of a rule cannot depend on more than a bounded portion of the top of the stack. Thus,

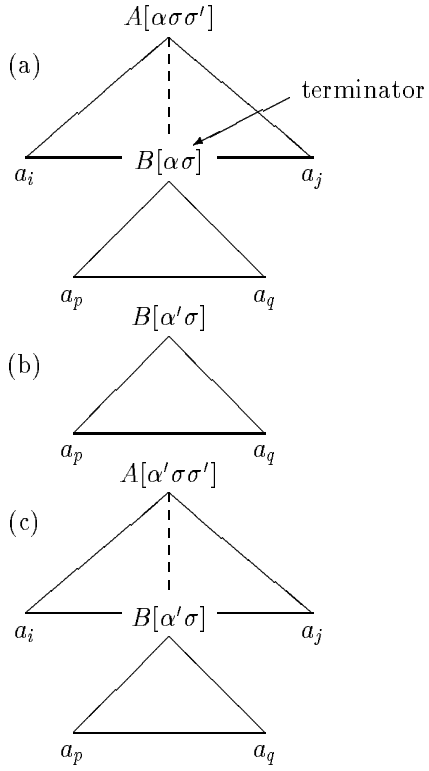


Figure 6: "Context-Freeness' in LIG derivations

rather than storing the whole of the potentially unbounded stack in a particular array entry, it suffices to store just a bounded portion together with a pointer to the residue.

Consider Figure 6. Tree (a) shows a LIG derivation of the substring $a_i \ldots a_j$ from the object $A[\alpha\sigma\sigma']$. In this derivation tree, the node labelled $B[\alpha\sigma]$ is a distinguished descendant of the root[2] and is the first point below $A[\alpha\sigma\sigma']$ at which the top symbol (σ) of the (unbounded) stack $\alpha\sigma$ is exposed. This node is called the *terminator* of the node labelled $A[\alpha\sigma]$. It is not difficult to show that only that portion of the derivation *below* the terminator node is dependent on more than the top of the stack $\alpha\sigma$. It follows that for any stack $\alpha'\sigma$, if there is a derivation of the substring $a_p \ldots a_q$ from $B[\alpha'\sigma]$ (see tree (b)), then there is a corresponding derivation of $a_i \ldots a_j$ from $A[\alpha'\sigma\sigma']$ (see tree (c)). This captures the sense in which LIG derivations exhibit "context-freeness". Efficient storage of stacks can therefore be achieved by storing in $P[i, j]$ just that bounded amount of information (nonterminal plus top of stack) relevant to rule application, together with a pointer to any entry in $P[p, q]$ representing a subderivation from an object $B[\alpha'\sigma]$.

Before describing how we adapt this technique to the case of PLPATR we discuss the sense in which PLPATR derivations exhibit a "context-freeness" property. The constraints on PLPATR which we have identified in this paper ensure that these feature values can be manipulated independently of one another and that they behave in a stack-like way. As a consequence, the storage technique used effectively for LIG recognition may be generalized to the case of PLPATR.

Suppose that we have the derived tree shown in Figure 7 where the nodes at the root of the subtrees τ_1 and τ_2 are the so-called f-terminator and g-terminator of the tree's root, respectively. Roughly speaking, the f-terminator of a node is the node from which it gets the value for the feature f. Because of the constraints on the form of PLPATR productions, the derivations between the root of τ and these terminators cannot in general depend on more than a bounded part of the feature structures $\boxed{1}$ and $\boxed{2}$. At the root of the figure the feature structures $\boxed{1}$ and $\boxed{2}$ have been expanded to show the extent of the dependency in this example. In this case, the value of the feature f in $\boxed{1}$ must be a, whereas, the feature g is not fixed. Furthermore, the value of the feature g in $\boxed{2}$ must be b, whereas, the feature f is not fixed. This means, for example, that the applicability of the productions used on the path from the root of τ_1 to the root of τ depends on the feature f in $\boxed{1}$ having the value a but does not depend on the value of the feature g in $\boxed{1}$. Note that in this tree the value of the feature g in $\boxed{1}$ is

$$F_1 = \left[\begin{array}{l} f : c \\ g : F_3 \end{array} \right]$$

and the value of the feature f in $\boxed{2}$ is

$$F_2 = \left[\begin{array}{l} f : F_4 \\ g : d \end{array} \right]$$

Suppose that, in addition to the tree shown in Figure 7 the grammar generates the pair of trees shown in Figure 8. Notice that while the feature structures at the root of τ_3 and τ_4 are not compatible with $\boxed{1}$ and $\boxed{2}$, they do agree with respect to those parts that are fully expanded at τ's root node. The "context-freeness" of PLPATR means that given the three trees shown in Figures 7 and 8 the tree shown in Figure 9 will also be generated by the grammar.

This gives us a means of efficiently storing the potentially unbounded feature structures associated

[2]The stack $\alpha\sigma$ associated with B is "inherited" from the stack associated with A at the root of the tree.

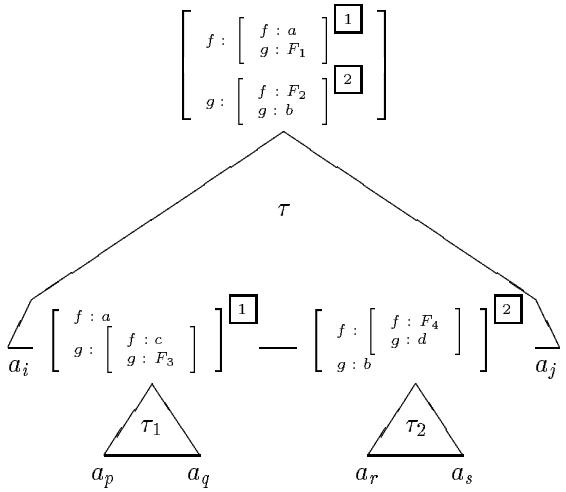


Figure 7: Terminators in PLPATR

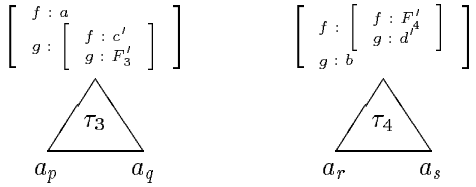


Figure 8: Compatible subderivations

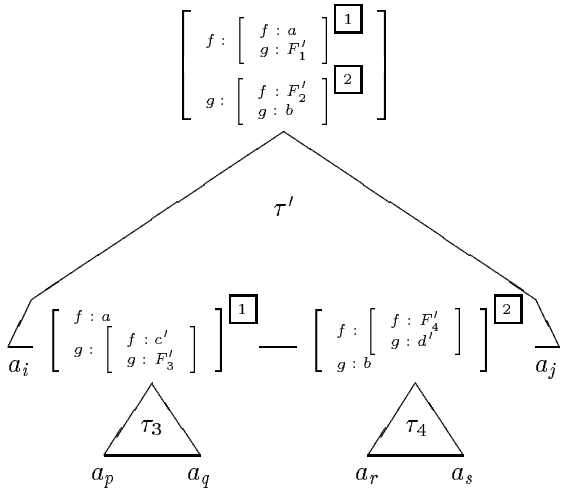


Figure 9: Alternative derivation

with nodes in a derivation tree (derived feature structures). By analogy with the situation for LIG, derived feature structures can be viewed as consisting of a bounded part (relevant to rule application) plus unbounded information about the values of features. For each feature, we store in the recognition array a bounded amount of information about its value locally, together with a pointer to a further array element. Entries in this element of the recognition array that are compatible (i.e. unifiable) with the bounded, local information correspond to different possible values for the feature. For example, we can use a single entry in the recognition array to store the fact that all of the feature structures that can appear at the root of the trees in Figure 9 derive the substring $a_i \ldots a_j$. This entry would be underspecified, for example, the value of feature $\boxed{1}$ would be specified to be any feature stored in the array entry for the substring $a_p \ldots a_q$ whose feature f had the value a.

However, this is not the end of the story. In contrast to LIG, PLPATR licenses structure sharing on the right hand side of productions. That is, partial linearity permits feature values to be shared between daughters where they are not also shared with the mother. But in that case, it appears that checking the applicability of a production at some point in a derivation must entail the comparison of structures of unbounded size. In fact, this is not so. The PLPATR recognition algorithm employs a second array (called the compatibility array), which encodes information about the compatibility of derived feature structures. Tuples of compatible derived feature structures are stored in the compatibility array using exactly the same approach used to store feature structures in the main recognition array. The presence of a tuple in the compatibility array (the indices of which encode which input substrings are spanned) indicates the existence of derivations of compatible feature structures. Due to the "context-freeness" of PLPATR, new entries can be added to the compatibility array in a bottom-up manner based on existing entries without the need to reconstruct complete feature structures.

6 Conclusions

In considering ways of extending LIG, this paper has introduced the notion of partial linearity and shown how it can be manifested in the form of a constrained unification-based grammar formalism. We have explored examples of the kinds of tree sets and string languages that this system can generate. We have also briefly outlined the sense in which partial lin-

earity gives rise to "context-freeness" in derivations and sketched how this can be exploited in order to obtain a tractable recognition algorithm.

7 Acknowledgements

We thank Roger Evans, Gerald Gazdar, Aravind Joshi, Bernard Lang, Fernando Pereira, Mark Steedman and K. Vijay-Shanker for their help.

References

Martin Emele. 1991. Unification with lazy non-redundant copying. In 29^{th} *meeting Assoc. Comput. Ling.*, pages 323—330, Berkeley, CA.

G. Gazdar. 1988. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69—94. D. Reidel, Dordrecht, Holland.

Dale Gerdemann. 1989. Using restrictions to optimize unification parsing. In *International Workshop of Parsing Technologies*, pages 8—17, Pittsburgh, PA.

Kurt Godden. 1990. Lazy unification. In 28^{th} *meeting Assoc. Comput. Ling.*, pages 180—187, Pittsburgh, PA.

S. P. Harrison and T. M. Ellison. 1992. Restriction and termination in parsing with feature-theoretic grammars. *Computational Linguistics*, 18(4):519—531.

L. Karttunen and M. Kay. 1985. Structure sharing with binary trees. In 23^{th} *meeting Assoc. Comput. Ling.*, pages 133—136.

T. Kasami. 1965. An efficient recognition and syntax algorithm for context-free languages. Technical Report AF-CRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Kiyoshi Kogure. 1990. Strategic lazy incremental copy graph unification. In 13^{th} *International Conference on Comput. Ling.*, pages 223—228, Helsinki.

F. C. N. Pereira. 1985. A structure-sharing representation for unification-based grammar formalisms. In 23^{th} *meeting Assoc. Comput. Ling.*, pages 137—144.

S. M. Shieber. 1984. The design of a computer language for linguistic information. In 10^{th} *International Conference on Comput. Ling.*, pages 363–366.

S. M. Shieber. 1985. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In 23^{rd} *meeting Assoc. Comput. Ling.*, pages 82–93.

Hideto Tomabechi. 1991. Quasi-destructive graph unification. In 29^{th} *meeting Assoc. Comput. Ling.*, pages 315—322, Berkeley, CA.

K. Vijay-Shanker and D. J. Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591—636.

K. Vijay-Shanker and D. J. Weir. 1994. The equivalence of four extensions of context-free grammars. *Math. Syst. Theory*, 27:511–546.

K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In 25^{th} *meeting Assoc. Comput. Ling.*, pages 104–111.

David Wroblewski. 1987. Nondestructive graph unification. In 6^{th} *National Conference on Artificial Intelligence*, pages 582—587, Seattle, WA.

D. H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.